Application Scalability and Performance on Multicore Architectures

Tyler A. Simon, Sam B. Cable, and Mahin Mahmoodi
USACE Engineer Research and Development Center, Major Shared Resource Center (ERDC MSRC), Vicksburg, MS
{Tyler.A.Simon, Sam.B.Cable, Mahin.Mahmoodi}@erdc.usace.army.mil

Abstract

The US Army Engineer Research and Development Center Major Shared Resource Center has recently upgraded its Cray XT3 system from single-core to dual-core AMD Opteron processors and has procured a quad-core XT4 supercomputer for installation in early 2008. This paper provides performance analysis of several representative Department of Defense applications executed on single-core and dual-core AMD Opteron processors. The authors provide a detailed strong-scaling study that focuses on addressing some areas of contention that may lead to increased job run times on applications running on many thousands of processors. The authors intend to use the results of this study as a guide for determining application performance on the quad-core Cray XT4.

1. Introduction

Commodity multicore chips have become an integral part of high performance computing architectures[1]. This paper examines the Cray XT3 at the US Army Engineer Research and Development Center Major Shared Resource Center (ERDC MSRC). With the move to multicore chips comes the expectation of improved code performance, as the number of computational threads is increased. With the ability of dual-core hardware to more efficiently handle multiple execution threads, the burden of increased code performance is placed primarily on the developers. This paper intends to help facilitate dialogue among users, developers, and hardware vendors in order to address the issues of application scalability and performance. This paper is primarily concerned with the performance of applications in use by the Department of Defense (DoD) user community.

2. System Overview

The current ERDC Cray XT3 (Sapphire) contains 4,160 processing nodes with each node running a 64-bit 2.6 GHz dual-core Opteron processor with 4GB dedicated memory. The nodes are connected to each other in a three-dimensional (3D) torus using a Hyper Transport link with a dedicated Cray SeaStar communications engine. Sapphire is rated at 42.6 TFLOP/s and contains 374TB of Fibre Channel RAID disk storage. Sapphire runs the UNICOS 1.5.39 operating system with the Catamount microkernel running on the compute nodes. Service nodes run a full SuSE Linux distribution with Cray XT3 extensions. The pre-upgrade system specifications included the 1.4.43 version of UNICOS and 4,096 nodes of 2.6 GHz AMD Opteron processors, with one core per node and 2GB of user accessible memory.

3. Benchmarks

Five codes were used in this study to evaluate the performance of the XT3. Four of the codes (AMR, GAMESS, LAMMPS, and OOCORE are a subset of the benchmarks that are used in the High Performance Computing Modernization Program (HPCMP) Technical Insertion (TI) procurement process, and also represent the HPCMP computational technology areas (CTAs). The fifth code, "HOMME", is not part of the TI-08 benchmarking package and is maintained by the National Science Foundation (NSF) and The National Center for Atmospheric Research (NCAR). Each code was executed with a fixed problem size on the single- and dual-core Sapphire nodes, with the dependent variable being run time. Each code was compiled with the PGI compilers with the default compiler optimization levels set "-02".

3.1. AMR

Properly speaking, "AMR" represents a collection of three adaptive mesh refinement codes developed and

0-7695-3088-5/07 $25.00 © 2007 IEEE

maintained by Lawrence Berkeley National Laboratory's Center for Computational Sciences and Engineering (CCSE) and the Applied Numerical Algorithms Group. This study used CCSE's "HyperClaw" code, which uses adaptive mesh refinement to solve hyperbolic systems of conservation laws[2,3]. The test case calculates solutions using the Navier-Stokes equations on a 3D grid with three levels of mesh refinement to simulate a gas dynamic shock impacting a helium bubble. Initially, the equations are solved in about 7 million cells. By the end of the simulation, this number grows to about 8 million. Figure 1 reflects the code run time as the number of cores increases, for single- and dual-core processors. The results in Figure 1 show good overall code scalability, but a slight increase in runtime can be seen with dual-core processors. This is likely due to the writing of 64, 25MB files per level of refinement, in this case 3. This results in a total output of around 60GB written throughout the run, including restart files. This input/output (I/O) behavior as well as the process distribution of the 3D mesh will be investigated in future work.

3.2. GAMESS

GAMESS is an *ab initio* molecular quantum chemistry code[4]. The GAMESS test case used in this study performed an MP2 computation that finds the nuclear gradient vector of a "BC4" molecule using a Restricted Hartree-Fock calculation with self-consistent field wave functions. GAMESS is a unique code, because half of the processes are tasked with computation and the other half handle inter-node communication. The majority of computation was spent computing energy integrals. The molecular data were calculated in the form of atom positions and electron orbitals. The I/O in this test case is representative of 80 percent of runs on Sapphire with occasional exceptions that may require 40GB or more. Figure 2 represents the scalability of the above test case on the single- and dual-core XT3. This version of GAMESS was compiled with all communication handled using Message Passing Interface (MPI). Figure 2 shows considerable improvement of GAMESS performance on the dual-core processors. With the single-core numbers, a performance slowdown at 1,024 processors is seen. This is due to the placement of computation and communication processes on nodes by the load-sharing facility of the scheduler. The dual-core performance increase is due to the round-robin striping mechanism that utilizes the improved on-chip, inter-process communication. This runtime variable "MPI_RANK_REORDER_METHOD" can be modified in the job submission script. In these tests computational and communication processes are placed on a single node with the dual-core processors, thereby reducing the overall communication latency. Thus, the dual process nature of GAMESS is exacerbated with multi-core processors. Future work in the analysis of this behavior will involve experimenting with multiple striping techniques to find optimal data placement.

3.3. HOMME

High Order Methods Modeling Environment (HOMME) was developed at the NCAR, which is operated by the University Corporation for Atmospheric Research. HOMME is a framework that provides the tools necessary to create a high-performance scalable global atmospheric model. HOMME supports execution on parallel computers using either MPI, OpenMP, or a combination of MPI/OpenMP.

HOMME is written in FORTRAN 90 and libraries in C. It uses Netcdf for formatting I/O; MPI for communication; and Lapack, Linpack, Blas and Metis for calculation and uses spectral elements. Spectral elements provide high-order accuracy while still maintaining a local communication pattern. Derivatives in spectral elements are implemented as small dense matrix-matrix operations. The test case used in this study is a 12-day Baroclinic instability simulation in 30 km spatial resolution in the horizontal dimension at equator (K = 13,824) spectral elements, (Np = 8) 64 points per spectral element, 96 vertical levels K*(Np-1)*(Np-1) = 677,376 horizontal grid points per vertical level maximum 13,824-way parallelism with ~1.3 TFlop hours of work. Figure 3 shows that HOMME scales very well and that running with dual-core processors has very little effect on run time. This is due to the fact that the HOMME jobs rely less on memory access than the other jobs.

3.4. LAMMPS

Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) is a classical molecular dynamics simulation code. It was developed at the Sandia National Laboratories and is maintained there. It is suitable for modeling parallel particles at the mesosacale or continuum level. The C++ version of LAMMPS was used in this study; because of class design of the code, it is easily extendable. The test case simulated a copper metallic solid with an embedded atom model (EAM) potential using a fixed-size problem with 108 million copper atoms on a lattice for 1,350 time-steps. The performance factors are flops-speed, memory bandwidth, and interconnect latency in the order of importance. Figure 4 shows a small reduction in scalability over 512 processors and significant improvement in dual-core run time. This performance gain is due to the increased access to CPU cycles, with reduced network

communication. This computationally intensive test case ran rather well on Sapphire.

3.5. OOCORE

OOCORE is a code developed by Oak Ridge National Laboratory, the University of Tennessee at Knoxville, and several other universities. OOCORE solves systems of linear equations that may be too large for the main memory of a set of CPUs to contain. In lieu of main memory, OOCORE stores the coefficient matrix in temporary files on the system's disk, so it generates a large amount of disk I/O. The OOCORE test case solved a linear system with 82,000 double complex unknowns with the LU factorization method. The systems were restricted to storing a relatively small maximum of 1.8 × 106 matrix elements in the memory of each processor; hence the calculation was always out-of-core on all the CPU counts. The performance factors are I/O capability and clock-rate. Figure 5 shows better dual-core performance for this I/O intensive code. The authors suspect this improvement is caused by the reduction in overhead caused by the Catamount microkernel. In this case, there are two processes per node, which reduces the relatively large disk latency imposed by system I/O. The effect of this I/O latency becomes more pronounced as cores are added. Scalability suffers perhaps more with OOCORE than with the other, more memory-dependent codes.

4. Conclusions

The dual-core performance of the AMD Opteron processors on the Cray XT3 performed well on some codes and increased run time on others. Figure 6 shows Sapphire's dual-core code execution time relative to the single-core runtimes. The horizontal line at 1.00 represents no performance gain, points below the line represent performance degradation and points above the line represent a performance improvement with dual core processors. The results show that on dual-core processors, two of the five benchmarks had runtimes at or near 90 percent of single core runtimes, with three codes running faster on dual core processors. The authors believe this is due to the following factors: process placement, off chip memory contention and parallel I/O. We suspect that quad-core performance will further fail to address these areas of contention and performance for these codes will slump further. Thus, solutions to poor multi-core performance might take the form of reducing network overhead by more logical process placement, updating or optimizing multi-node I/O, and working with more distributed- and shared-memory programming models. In order to get any performance gain from large scientific applications in a multi-core environment we should focus on mapping application processes to cores at runtime, and reduce areas of core contention. Explicit processes can be monitored and manipulated at the operating system level, with compiler flags, with the job scheduler, and at the MPI level during runtime. We recommend starting with these areas first, then examining more application specific work placement to get optimal performance from codes. These factors will be examined in more detail in future work.

Acknowledgments

This work was conducted using computer time from the DoD High Performance Computing Modernization Program at the ERDC MSRC, Information Technology Laboratory, Vicksburg, MS.

References

1. Aitken, P. et al., "Thriving and Surviving in a Multi-Core World, AMD eBook from AMD Developer Central." November 2006, http://developer.amd.com/assets/ThrivingandSurvivinginaMulti-CoreWorld.pdf.

2. Rendleman, Charles A., et al., "Parallelization of Structured, Hierarchical Adaptive Mesh Refinement Algorithms." Lawrence Berkeley National Laboratory, Berkeley, CA, 1999, http://seesar.lbl.gov/CCSE/ Publications/car/ParHyper.pdf.

3. http://seesar.lbl.gov/CCSE/Software/index.html and http://seesar.lbl.gov/AMR/index.html.

4. Schmidt, M.W., K.K. Baldridge, J.A. Boatz, S.T. Elbert, M.S. Gordon, J.H. Jensen, S. Koseki, N. Matsunaga, K.A. Nguyen, S. Su, T.L. Windus, M. Dupuis, and J.A. Montgomery, "General Atomic and Molecular Electronic Structure System." *J. Comput. Chem.*, 14, pp. 1347–1363, 1993.

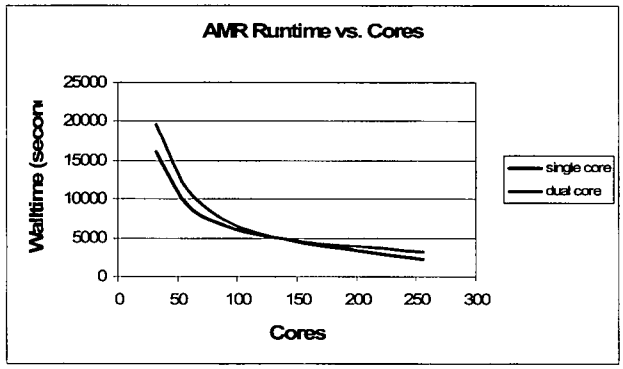

Figure 1. AMR run time scalability

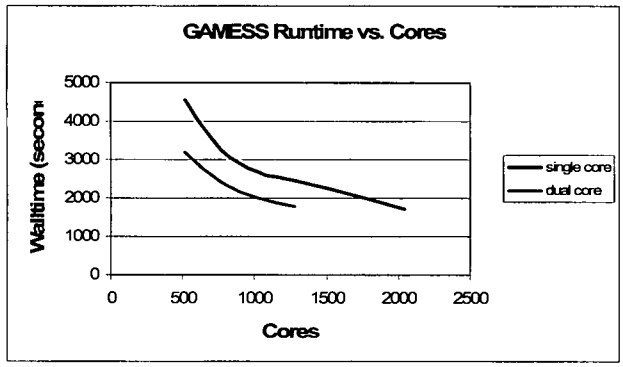

Figure 2. GAMESS run time scalability

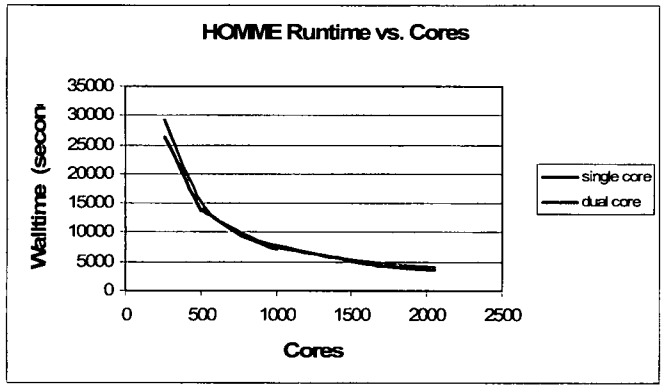

Figure 3. HOMME run time scalability

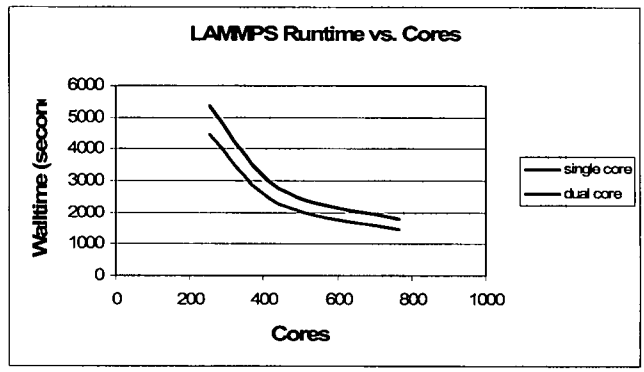

Figure 4. LAMMPS run time scalability

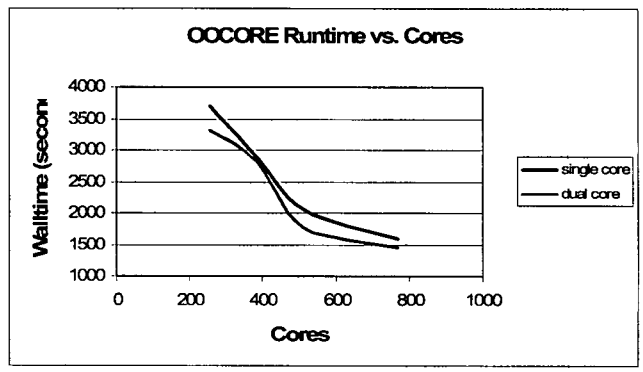

Figure 5. OOCORE run time scalability

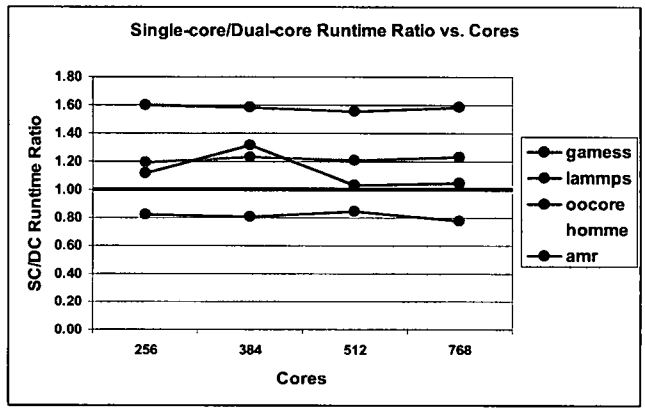

Figure 6. Single-core vs. dual-core speedup